CQSIM

High-Level Design Document

Ren Dongxu

1. OVERALL

1.1 About the Manual

- This manual is focus on how to use and extend the simulator.
- This manual will not go too deeper into the details of the implementation. It will only show you the structure to help you understand CQSIM. Read the design document if you want to know more.
- If you just want to know about the CQSIM structure or how these modules work together, jump to **3. STRUCTURE**.

1.2 About the CQSIM

- CQSIM is formed by several modules, each one is implemented as a class. Most of them follow the Object-Oriented Programming.
- Some data are supposed to know by all modules. This violates the Object-Oriented Programming, but it makes the simulator more efficient.
- This version is focus on the SWF file simulating. You may need to rewrite some module to make it fit other types of job trace.

2. FEATURES

- The section shows you the principles when designing the simulator structure.
- You may need to modify all the classes at last. But keep the structure good will always help the simulator growing in the least cost.

2.1 Reusability

- Make the modules can handle more situation without rewriting them.

2.1.1 Config file

Most modules have their own config files. The values of parameters are set in the config file so that user need not rewrite the source code only to modify the parameters.

2.1.2 Default Input

Most input parameters have their default value. It also guarantee the backward compatibility.

2.1.3 Extendable Input

Some parameters are defined as a dictionary or list. So that user can extend the input parameters while keeping the interface unchanging.

2.2 Extensibility

- Most modules are independent from each other. So user can rewrite the single class to extend the function.
- For example:
  The **Node Structure** module takes care of all node operations. The other modules need not to know what the node structure topology is, they only need to send the node requirement to the module and ask "Is this operation available?"
  * For more detail, see below section.

2.3 Efficiency

- To implement efficiency, some functions are not used and some rules have to be broken.
- Efficiency is a trade-off towards Reusability and Extensibility. Sometimes Reusability and Extensibility are very expensive, in both simulator running time and project maintain time.
- Job information format is supposed to know by all modules, the details of **Info_collect** class is supposed to know by all modules. Also, node allocation tracking function is not used to highly improve the running speed. More details will be provided below.

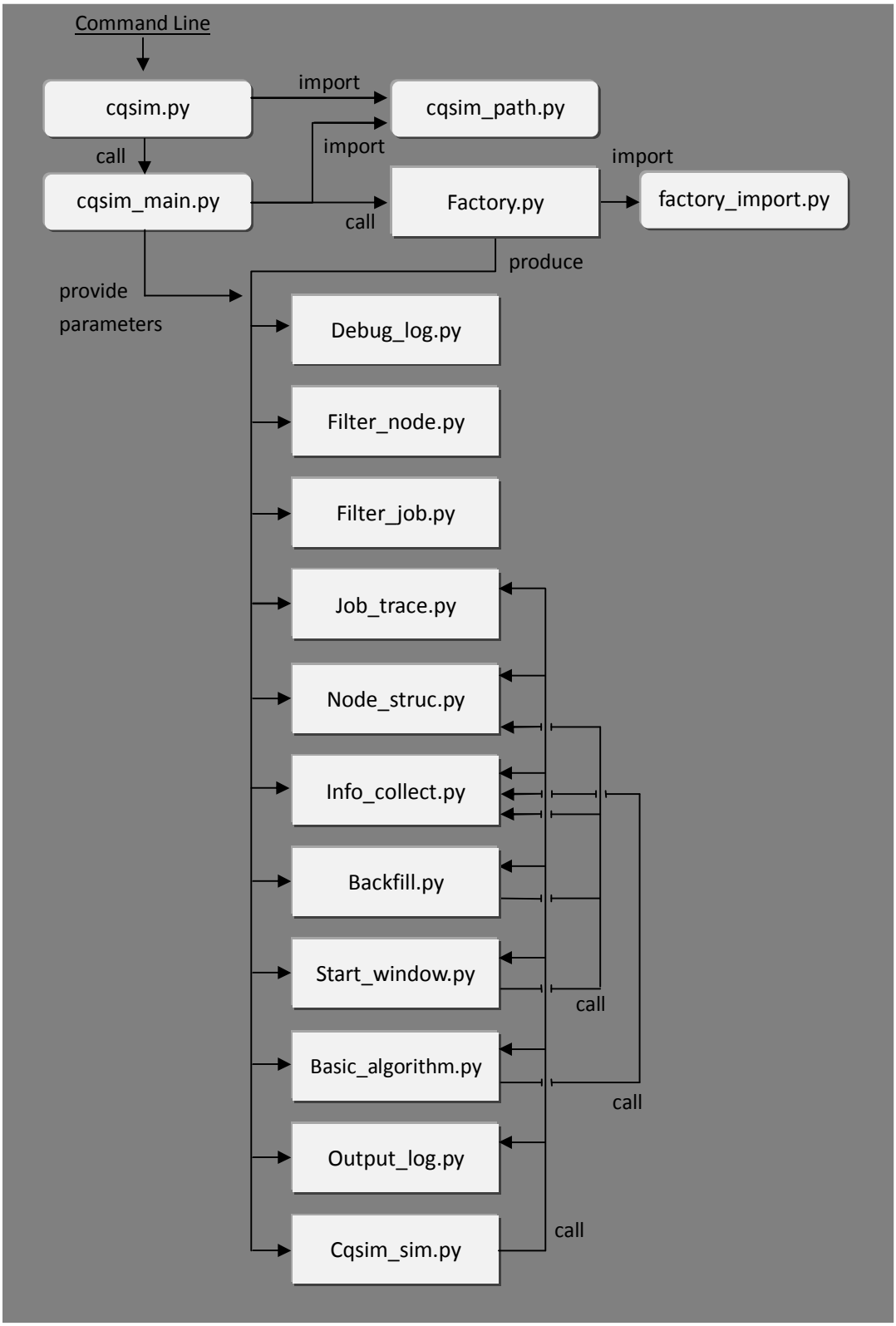
3. STRUCTURE

3.1 Files Deployment

- __init__.py files are used to make Python treat the directories as containing packages. Not showing in the table.
- *.pyc files are not showing in the table.
- **Borden** with dark back-color items are folders.

Cqsim	**data**	**Debug**	…		debug log folder
		Fmt	…		formatted files folder
		InputFiles	…		original input files folder
		Results	…		result log folder
	documents	*<Empty>*			documents place
	src	cqsim.py			command line user interface
		cqsim_ad.py			advantage command user interface, not implemented
		cqsim_main.py			call and initialize every module with the input, then run the simulator
		cqsim_path.py			contains the path variables, change it if you change the file deployment
		Factory.py			module factory
		factory_import.py			forms the data of module groups
		Config	ad_alg_para.set		Basic Algorithm adapt parameter config file
			ad_bf_para.set		Backfill adapt parameter config file
			ad_win_para.set		Start Window adapt parameter config file
			config_n.set		log names config file
			config_sys.set		system config file
		CqSim	Backfill.py		Backfill module
			Basic_algorithm.py		Basic Algorithm module
			Cqsim_sim.py		the simulator control center, control all modules
			Info_collect.py		Information Collect module
			Job_trace.py		Job Trace module
			Node_struc.py		Node Structure module
			Start_window.py		Start Window module
		Extend	**SWF**	Filter_job_SWF.py	these are the SWF version modules.
				Filter_node_SWF.py	
				Node_struc_SWF.py	
		Filter	Filter_job.py		Job Filter module
			Filter_node.py		Node Structure Filter module
		Interface	*<Empty>*		
		IOModule	Debug_log.py		Debug Output module
			Log_print.py		Log Print module, invoked by all output module
			Output_log.py		Result Log Output module
		Old Version	*<Not Important>*		contains some old files, not used now, some codes in them may be useful
		ResultAnalysis	*<Empty>*		

3.2 Brief Interact Chart



- The flow is:
  - cqsim.py receive all input parameter from command line or config files.
  - cqsim_main.py instantiates all modules by factory mode, and pass the parameters to the

corresponding module.

- Modules are initialized in order. At last, the handles of 7 modules are passed into Cqsim_sim and start simulating.

- Log_print module (not show in the chart) is the only one that does not instantiate in cqsim_main.py.

4. MODULES & FUNCTIONS

4.1 cqsim

File name	cqsim.py
Description	• Command line interface. Receive parameters, if no parameters input, it will read the config files(system config file and name config file in order). • It collects the data and pass them to **cqsim_main**

4.2 cqsim_path

File name	cqsim_path.py
Description	• This file contains and only contains path variables. • For example: The command line only gives the original job trace file name ”a.swf”, **cqsim_path** provides the corresponding path variable “../data/Input/” adding to the head of the job trace name.

4.3 cqsim_main

File name	cqsim_main.py
Description	• Receive the parameters from **cqsim**. • Call the **Factory** to produce every module, and transfers corresponding parameters to them. • The running time of the simulator is got from this method. • Be careful to change the initializing order of the modules, some of them need to be initialized after the others. • It can be separated into 4 sub step: 1. Initialize the **Debug** module, which will be call by every other module. 2. Formats the original data by **Job Filter** and **Node Filter**, then initialize the **Job Trace** and **Node Structure** module with the formatted data. 3. Initialize **Information Collect**, **Backfill**, **Basic Algorithm**, **Start Window** and **Result Log** module. 4. Pass the handles of the modules into **Cqsim Control** module and run the simulator.

4.4 Factory

File name	Factory.py factory_import.py
Description	• Factory mode. In order to organize and manage all version of modules. • Factory data is separated from the factory class, storing in the file factory_import.py. So, you can modify the data only, the factory class always remains the same. Or you can even create the factory data dynamic, if you just store all information into a database and just pick the right module group when you run the simulator.

4.5 Debug Log

File name	Debug_log.py
Description	• Output the debug information to the debug log file. • You can insert the debug command into the code where you want to track some values. • You can also adjust the debug level to make the debug context show on the console in running time. • Do not keep too much debug command in the code, every command need a file open/write/close operation group. May make the speed down.

4.6 Filters

File name	Filter_node.py Filter_job.py
Description	• Format the job and node data. • Produce a formatted job file and a formatted node structure file. • Also produce a job config file and node structure config file.

4.7 Job Trace

File name	Job_trace.py
Description	• Keep all formatted job data • Keep some overall job trace information *e.g. job trace start date* • Provide some modification on job data *e.g. modify the submit time* • Provide all job actions, which are well packed. *e.g. other module only need to call submit() when job submit, all the relating data operations are done by the submit()*

4.8 Node Structure

File name	Node_struc.py
Description	• Keep all formatted node information *e.g. idle node number, total node number* • Provide all node actions, which are well packed. • In order to make the simulator efficient, the node list will not be updated in running time. Instead, some variables is used to indicate the overall node situation. • The node request is a dictionary, that means you can add new node requirement unit. *e.g. In current version, the dictionary contain only an item "proc", you can also add new item "node", "core" for more complex node topology.*

4.9 Information Collect

File name	Info_collect.py
Description	• Collect all running time information rather than job/node. • These information are used in 2 ways:

	(1). for report (2). for information analysis, used by the adapt functions. • **Backfill**, **Start Window** and **Basic Algorithm** can "break into" this module and add the average interval time they need. • This module is called in 2 situations: (1). job submit/start/finish (2). monitor event • So, if you want to add a new data to collect, you need to add the data name, and the methods which be called to get the value in these 2 cases. Then add the item into the data list, so that the new methods will be called automatically.

4.10 Backfill

File name	Backfill.py
Description	• When the next job (ordered by scores) can not be started and there are more than 1 job in waiting list, **Backfill** module will be called to check whether there is any job can be backfilled according to the backfill rule. • This module will only return the backfill job index list. It will not start the job. • You can add new backfill method and allocate a mode number to it. • **Backfill** module supports adapt function, which is changing the parameters in running time according to different system state.

4.11 Start Window

File name	Start_window.py
Description	• Just after a job submit or finish and before any job get started, the simulator will call **Start Window** module to adjust the first x jobs to get a local greedy order. And, assuming there are more than x jobs in the waiting list and the first x jobs all get started, the **Start Window** module will be called again to reorder the next x jobs. • 3 number (let's say, *a*, *b*, *c*)will need to be input: **Start Window** module will scan first *a* jobs in the waiting list, and try all the arrangement of the first *b*(*b* <= *a*) jobs to get a best solution, then return the new order. And, after *c* jobs get started on the same time point, the **Start Window** module will be called again to scan the next *a* jobs. • **Start Window** module supports adapt function.

4.12 Basic Algorithm

File name	Basic_algorithm.py
Description	• After any job submit/finish event, **Basic Algorithm** module will be called to calculate the scores for every job in the waiting list, and reorder them. • Algorithm string is formed by input algorithm elements and be used to calculate the score though the eval() function.

	• **Basic algorithm** module supports adapt function.

4.13 Result Log

File name	Output_log.py
Description	• This module can be very frequent modified. Because you may need the simulator output different data, or in different type even in the same experiment. • Output 3kinds of result logs. • System information log: Contains running time system state, ordered by time order. Print when **Information Collect** module is called • Job result log: Contains all job information, ordered by job index Print when all jobs are done. • Adapt information log: Contains the adapt information Print when adapt function happens.

4.14 Cqsim Controller

File name	Cqsim_sim.py
Description	• This is the controller of the whole simulator. • It contains the event sequence • After add all job submit events into the sequence in time order, simulator begin running by move from one event to the next. • Monitor events, job end events will add to the sequence at the right place. • After job submit/end events, simulator try to start the jobs in the waiting list. • When all events in the sequence are done, simulating is done.

5. DETAILS

- This is an additional explanation to show how the functions work.

5.1 Basic Algorithm

(1). Algorithm elements are input into **Basic Algorithm** module in a list when **Basic Algorithm** module is initialized

e.g. we input the list ['w','','t','^^','3'], we will use the example to go through this section.*

(2). **Basic Algorithm** module groups the elements together and builds the algorithm string.

e.g. algorithm string = 'w'+''+'t'+'^^'+'3' = "w*t^^3"*

(3). Define the local variables, in order to make the "letter" in the algorithm string stands for one of the job information.

s	job submit time
t	job estimated time
n	job required nodes #
w	job waiting time
m	current idle nodes #

*e.g. algorithm string "w*t^^3" means* ***job waiting time * job estimated time^3***

(4). Make the algorithm string works as an expression by using eval()

e.g. ***x=eval("w*t^^3")*** *is the same as* ***x= w*t^^3***

(5) Now we get the score.

5.2 Adapt Function

5.2.1 Structure

- Adapt function can be simplified in the following way:

  a (e.g. window size in ***Start Window*** *module)* is the variable need to be modified in running time.

  b (e.g. average utilization in last 30 minutes) and *c (e.g. average utilization in last 10 hours)* are the system information which influence *a*.

  So the adapt function is:

$b > c$	$a = a_1$
$b < c$	$a = a_2$

- So, we only need to tell simulator which parameter is *a*, which are *c* and *b*. And give the cases. Then make simulator automatically calculate *b* and *c*, check which case fits the system state and modified parameter *a* at last. This is the total idea of the adapt function.
- There are 2 points to implement the adapt system:
  - Parameter *a* should be stored in a dictionary, so simulator can find it by provided name.
  - The **Information Collect** module should automatically calculate the system information *b* and *c*. So, the adapt modules should "break into" **Information Collect** module, and add the method of getting *b* and *c*.

5.2.2 Adapt Config file

- This config file contains the following information:

Name	Comment

adapt_data_name	Adapt data name in the dictionary.
adapt_data_para	Adapt data parameter, provide the index if the adapt data is a list
check_data_name	The name of data need to be check when adapt
check_data_para	check data parameter.
avg_uti	Average utilization interval list. This list contain all the average utilization need to be check
adapt_item	adapt case
bound_item	bound for every adapt data.

5.2.3 Initialize

(1). Module reads its own adapt config file.

(2). Module sends its interval list to **Information Collect** module, the **Information Collect** module will add the interval times and return the list of the index of the interval times. The returned index list will be used to get the average data in running time.

(3). When all the initial works are done, **Information Collect** module will produce a order list. This order list reorders the interval times from short to long.

5.2.4 Run

- The adapt methods will be called in monitor event by **Cqsim Controller** module.
- In the adapt method, every adapt case will be checked in order, until one fits the current state or no more case left.

6. EXTENSION

- There are 3 ways to extend the simulator

6.1 Inherit

- Create a new class inheriting the super class.
- The same functions can be reused.
- The interface remains same, so the other modules do not need to know the changing. They also need not to be modified.
- Extend the module in this way when the details of the old version and new version are different from each other and the other modules do not care about the differences.
- **Job Filter**, **Node Structure Filter**, **Job Trace**, **Node Structure** modules are extended in this way.

6.2 Add Method

- You can just add a new method to the module without creating a new class.
- In this way, all old functions remain same. New function is added, but you can choose not to use it.
- This kind of extension makes the simulator provide more choose for the user.
  e.g. You can add a new backfill mode other than EASY and conservative an allocate mode number 2 to it. So user can use the new backfill function by input mode number 2.
- **Backfill**, **Basic Algorithm**, **Start Window** modules can be extended in this way.

6.3 Modify Old Code

- Sometimes you will find the old code can not support new function.
- This kind of extension may affect the whole simulator. So you need to be careful and make sure the extension is a general extension, not just a special case.
- If it is a special case, just create a new class and extend it in the first way.